# Universidad Carlos III

## Heurística y Optimización

## 2023-24

## Práctica 2


Ingeniería Informática, Tercer Curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)

César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)

Prof. María Fernanda Salerno Garmendia

Grupo: 81

# Índice

# 1. Introducción

En este documento se recoge el desarrollo de la práctica 2 de Heurística y Optimización. La práctica consiste en resolver dos problemas, uno de satisfacción de restricciones y otro de búsqueda.

El primer problema consiste en la asignación de plazas a una serie de ambulancias bajo ciertas condiciones, que se detallarán más adelante. El segundo problema consiste en encontrar el camino óptimo para recoger a un número de pacientes contagiosos y no contagiosos y llevarlos a los centros sanitarios correspondientes. Para el primer problema nos hemos valido de la biblioteca *constraint* de python, mientras que para el segundo hemos implementado el algoritmo *a** en C++.

A continuación se incluye la descripción del modelo y el análisis de los resultados para cada parte. Además de conclusiones de la práctica.

# 2. Descripción del modelo

## 2.1. Modelo Parte 1

El problema consiste en asignar a cada ambulancia una plaza de un parking teniendo en cuenta distintas restricciones.
Existen ambulancias con congelador y sin congelador. Además, cada ambulancia puede ser de servicio urgente (TSU) y servicio no urgente (TNU). En el parking existen plazas con electricidad y sin electricidad.
Tendiendo todo esto en cuenta, el modelo se corresponde con una red de restricciones de la forma $R = (X, D, C)$ donde:

- X es el conjunto de variables. Cada ambulancia será una variable. Las variables se pueden agrupar en cuatro grupos: tsu con congelador (TSU-C), tsu sin congelador (TSU-X), tnu con congelador (TNU-C) y tnu sin congelador (TNU-X).
- D es el conjunto de dominios. Las ambulancias con congelador tendrán el dominio de las plazas con electricidad, mientras que las demás tendrán el dominio de cualquiera de las posiciones del parking. Una plaza será identificada por un par ordenado de coordenadas *(x, y)*.

$$D_{TSU-C} = D_{TNU-C} = \{(x, y) : (x, y) \in PlazasElectricas\}$$

$$D_{TSU-X} = D_{TNU-X} = \{(x, y) : 1 \leq x \leq xSize \wedge 1 \leq y \leq ySize\}$$

  Donde *xSize* y *ySize* es el tamaño máximo del parking en cada una de las dimensiones.
- C es el conjunto de restricciones, que detallamos a continuación.

**Restricciones**
El enunciado detalla las siguientes restricciones:

- Toda ambulancia debe tener una plaza de aparcamiento asignada: esta restricción viene garantizada por el dominio. Para que exista una solución, todas las variables deben tener un valor asignado dentro de su dominio.

- Dos ambulancias no pueden tener asignada la misma plaza de aparcamiento:

$$R_{ij}{}^{1} = \{((x_i, y_i), (x_j, y_j)) : x_i \neq x_j, y_i \neq y_j\}$$

- Una ambulancia provista de congelador sólo podrá ocupar una plaza con electricidad. De nuevo, vendrá garantizada por el dominio de este tipo de ambulancias.
- Una ambulancia debe tener siempre una plaza libre al lado, ya sea a la derecha o a la izquierda. Esto se ha modelado de dos formas: en general, 3 ambulancias no pueden ser consecutivas en la misma columna; y, si una ambulancia está junto a un borde (superior o inferior), no puede estar consecutiva a otra ambulancia en su misma columna.

$$R_{ijk}{}^{2} = \{((x_i, y_i), (x_j, y_j), (x_k, y_k)) : y_i \neq y_j \neq y_k \vee x_i \neq x_j + 1 \neq x_k + 2\}$$

$$R_{ij}{}^{3} = \{((x_i, y_i), (x_j, y_j)) : (x_i \neq 1 \wedge x_i \neq m) \vee (y_i \neq y_j \vee x_i \neq x_j + 1)\}$$

Donde $m$ es el valor máximo de $x$ en el parking.

**Implementación**

El modelo se ha implementado haciendo uso del módulo *constraints* de Python. Se han desarrollado dos módulos, uno encargado de la lectura y escritura y otro para la inicialización y obtención de soluciones del problema.

Para el archivo de entrada, se ha asumido que todas las líneas acaban con un "\n".

Para el archivo de salida, se imprimen 3 soluciones aleatoriamente o, si existen menos de 3, se imprimen todas las soluciones. Cada solución vendrá separada por una línea de asteriscos.

## 2.2. Modelo Parte 2

La segunda parte consiste en un problema de búsqueda del camino óptimo para recoger una serie de pacientes y llevarlos a los centros de atención cumpliendo ciertas normas.

De esta forma, nuestro problema estará caracterizado por un estado, una serie de operadores, un conjunto de estados iniciales y un conjunto de estados finales.

Además, existen algunas constantes del problema como son el mapa, la capacidad de la ambulancia o la energía máxima de la ambulancia.

**Estado**

El estado incluye los siguientes campos:

- Ambulancia:
  - Posición: dos valores x e y que especifican su posición en el mapa.
  - Energía: determina el coste del camino que puede recorrer antes de volver al parking.
  - Enfermos contagiosos: número de enfermos contagiosos (C) que carga la ambulancia.
  - Enfermos no contagiosos: número de enfermos no contagiosos (N) que carga la ambulancia.

    - Posición enfermos contagiosos: lista de las posiciones de enfermos contagiosos(C)
    - Posición enfermos no contagiosos: lista de las posiciones de enfermos no contagiosos (NC)
- Contagiosos: número de enfermos contagiosos que no se han llevado a su centro sanitario.
- No Contagiosos: número de enfermos no contagiosos que no se han llevado a su centro sanitario.

**Implementación**

El modelo se ha implementado a través de distintas clases en C++. Hemos elegido este lenguaje de programación por la velocidad que presenta frente a alternativas como Python.

Para la implementación, hemos implementado un heap propio para usarlo como lista abierta y hemos usado un *unordered_map* como lista cerrada. Debido a que la inserción en el heap puede ser lineal, este es el principal cuello de botella en la ejecución del algoritmo.

Además, tal y como pide el enunciado, se ha desarrollado un script de Python *ASTARTraslados.py* que permite compilar y ejecutar el archivo a través del siguiente comando.

```
python ASTARTraslados.py ASTAR-tests/<mapa> <h>
```

Donde *h* es un número del 1 al 5.

**Operadores**

Para mayor claridad, definiremos la función *consecuences*, encargada de de evaluar la casilla sobre la que se encuentra la ambulancia y provocar los efectos adecuados.

```
consecuences(casilla) {

if casilla == parking then ambulancia.energy = 50

if casilla == hospital_c then contagiosos -= ambulance.contagiosos, ambulancia.contagiosos = 0

if casilla == hospital_nc && ambulancia.contagioso == 0 then no_contagiosos -= ambulancia.no_contagioso, ambulancia.no_contagioso = 0

if casilla == contagioso && ambulancia.contagioso < 2 && ambulancia.no_contagioso <= 8 then ambulancia.contagiosos + 1, ambulancia.pos_contagiosos - contagioso // Se elimina el contagioso de la lista

if casilla == no_contagioso && ambulancia.contagioso == 0 && ambulance.no_contagiosos < 10 then ambulancia.no_contagiosos + 1, ambulancia.pos_no_contagiosos - no_contagiosos // Se elimina el no contagioso de la lista}
```

Los operadores permiten transicionar entre estados, existen los siguientes operadores:

- *move_right:* se desplaza la ambulancia a la casilla inmediatamente a la derecha si es que esa casilla está permitida y tiene suficiente energía.

  ```
  IF(ambulancia.y + 1 <= limit_y && ambulancia.energy > casilla.coste)
  THEN  ambulancia.y = ambulancia.y + 1 && ambulancia.energy - casilla.coste,
  consecuences(ambulancia.get_casilla)
  ```

- *move_left:* de manera similar a move_right, pero para el desplazamiento hacia la izquierda y tiene suficiente energía.

  ```
  IF(ambulancia.y - 1 <= limit_y && ambulancia.energy > casilla.coste)
  THEN  ambulancia.y = ambulancia.y - 1 && ambulancia.energy - casilla.coste,
  consecuences(ambulancia.get_casilla)
  ```

- *move_up:* de manera similar a las anteriores, pero para el desplazamiento hacia arriba

  ```
  IF(ambulancia.x - 1 <= limit_y && ambulancia.energy > casilla.coste)
  THEN  ambulancia.y = ambulancia.x - 1 && ambulancia.energy - casilla.coste,
  consecuences(ambulancia.get_casilla)
  ```

- *move_down:* de manera similar a las anteriores, pero para el desplazamiento hacia abajo

  ```
  IF(ambulancia.x - 1 <= limit_y && ambulancia.energy > casilla.coste)
  THEN  ambulancia.y = ambulancia.x - 1 && ambulancia.energy -
  casilla.coste, consecuences(ambulancia.get_casilla)
  ```

# 3. Análisis de los resultados

## 3.1 Análisis Parte 1

Para validar el programa, se han definido 7 casos, además del caso del enunciado, que validan cada una de las restricciones de forma individual. Para cada uno se han especificado las soluciones posibles de forma que podamos validar el número de soluciones obtenidas.
Se pueden ejecutar todas los casos de prueba al ejecutar el archivo *CSP-calls.sh* desde la carpeta parte1/.

### Resumen de los casos de prueba:

| Archivo | Soluciones esperadas | Soluciones obtenidas |
|---|---|---|
| **parking01-feasible.txt** | 2 | 2 |
| **parking02-no_feasible.txt** | 0 | 0 |
| **parking03-feasible.txt** | 4 | 4 |

| | | |
|---|---|---|
| **parking04-no_feasible.txt** | 0 | 0 |
| **parking05-feasible.txt** | 2 | 2 |
| **parking06-feasible.txt** | 6 | 6 |
| **parking07-no_feasible** | 0 | 0 |
| **parking56.txt** | - | 2.175.288 |

*Para el último caso no se conocen las soluciones esperadas.*

**Parking01-feasible:**
Se trata de un parking de 4 (2x2) plazas, en las que dos cuentan con electricidad y están colocadas en diagonal. De esta forma, la restricción más limitante es la impuesta sobre ambulancias con congelador y plazas eléctricas. Sólo existirán dos soluciones.

**Parking02-no_feasible:**
Se trata del mismo parking que el caso anterior. Ahora existirán 3 ambulancias con congelador, de forma que es imposible colocar toda ambulancia con congelador en plazas eléctricas.

**Parking03-feasible:**
Se trata de un parking de una única columna y cinco filas, con una única plaza eléctrica y tres ambulancias (una de ellas con congelador). De esta forma, la posición de la ambulancia con congelador es fija, y las otras dos se deben distribuir de forma que dejen una plaza libre entre ellas. Con esta configuración, existen 2 distribuciones posibles; además, como las ambulancias sin congelador pueden intercambiarse el número de soluciones se duplica, teniendo un total de 4 soluciones.

**Parking04-no_feasible:**
Se trata de un caso similar al anterior, con una fila menos y ambulancias sin congelador. Es fácil comprobar que no existe distribución de las ambulancias posibles sin que se incumplan las restricciones $R_{ijk}{}^{2}$ y $R_{ij}{}^{3}$.

**Parking05-feasible:**
Se trata de un parking 4x1 con dos ambulancias, una de ellas con posición fija junto al borde superior. De esta forma comprobamos la restricción $R_{ij}{}^{3}$, existiendo sólo dos soluciones posibles.

**Parking06-feasible:**
Se trata de un parking de 4 (2x2) plazas y dos ambulancias (una TNU y una TSU). De esta forma probaremos la restricción $R_{ij}{}^{1}$. Es fácil comprobar que sólo existen 6 soluciones posibles: 4 posibilidades al colocar las ambulancias en diagonal y 2 soluciones donde la TSU se coloca inmediatamente delante de la TNU.

**Parking07-no_feasible:**
Se trata de un parking 2x3 en el que existen 2 ambulancias TSU-C y una ambulancia TNU-X, de forma que las TSU's tienen su posición fija lo más a la izquierda posible. De esta forma, la

ambulancia TNU-X sólo tendrá disponibles plazas delante de las TSU. En ese caso, se incumple la restricción $R_{ij}{}^{1}$. Por lo que no existen soluciones.

**Parking56**

Este es el parking facilitado como ejemplo en el enunciado. Se ha utilizado como prueba de esfuerzo y para comparación del número de soluciones con las guías facilitadas por la profesora. Para este problema, obtenemos en torno a las 2 millones de soluciones (2175288 soluciones).

## Conclusiones

Tras validar cada una de las restricciones de forma individual, concluimos que nuestro modelo resuelve adecuadamente el problema planteado.

## 3.2 Análisis Parte 2

**Casos de prueba**

Para probar la implementación del A* y las heurísticas implementadas vamos a utilizar los siguientes mapas.

- **Mapa sencillo 3x3**: Este mapa nos permitirá probar de forma rápida que el A* funciona correctamente. Se podrán comprobar características básicas como que se reduce la gasolina de la ambulancia al moverse o que recoge y suelta a los enfermos en el orden correspondiente.
- **Mapa con parking encerrado**: En este mapa el parking se encuentra encerrado entre casillas intransitables, por lo tanto es imposible que se encuentre una solución.
- **Mapa con mucho coste**: Este mapa es una mapa reducido (5x5) en el que todas las casillas que no tienen elementos son de coste 2, por lo tanto la ambulancia se verá obligada a volver al parking para recargar la energía.
- **Mapa con solo contagiosos**: En este mapa solo se dispondrá de contagiosos.
- **Mapa con solo no contagiosos**: En este mapa solo se dispondrá de no contagiosos.
- **Mapa con contagioso obligatorio:** Este mapa tendrá el parking encerrado por contagiosos. En este caso veremos si las prioridades de los enfermos se están aplicando correctamente, ya que al tomar a un contagioso de forma obligatoria no podrá tomar ningún no contagioso hasta soltarlo.
- **Mapa recta**: Este mapa nos permite comprobar de forma rápida si las heurísticas son admisibles, ya que el coste es trivial.
- **Mapa con enfermo alcanzable pero no se puede volver**: En este mapa se tendrá a un enfermo en una esquina y sus casillas adyacentes tendrán coste 26, de esta manera se podrá entrar en la casilla del enfermo para recogerlo pero no se podrá volver. Este mapa no tendrá solución por lo tanto.
- **Mapa de estrés**: Este mapa será grande para probar la calidad de la heurísticas, ya que el tiempo que llevará encontrar la solución óptima sin ellas será muy grande.

Se pueden ejecutar todos los casos de prueba salvo el *mapa de estrés* (por motivos de tiempo de ejecución para algunas heurísticas) ejecutando el script *ASTAR-calls.sh* desde la carpeta parte2/.

**Heurísticas Propuestas**

Hemos propuesto 5 heurísticas, de las primera de ellas han surgido las restantes excepto la última

- **H1**: Esta heurística consta de dos partes, si no quedan ningún enfermo por entregar y si queda algún enfermo por entregar
    - Si no queda ningún enfermo por entregar la heurística devolverá el valor de la distancia entre la posición de la ambulancia y el parking
    - Si queda algún enfermo por entregar la heurística devolverá el valor de la distancia al enfermo más lejano a la ambulancia más la suma de la distancia de su correspondiente hospital. Si en un momento en concreto el enfermo más lejano es un no contagioso se devolverá la suma de la distancia a este enfermo y la distancia entre este enfermo al hospital de no contagiosos.
- **H2**: Esta heurística consta de varias ramas según la situación actual de la ambulancia.
    - Si aún quedan enfermos por entregar se tendrá 3 posibilidades
        - Si la ambulancia lleva a exactamente 1 contagioso, la heurística devolverá la distancia al contagioso más lejano más la distancia de ese contagioso al hospital de contagiosos
        - Si la ambulancia lleva a exactamente 2 contagiosos, la heurística devolverá la distancia de la ambulancia al hospital de contagiosos
        - Para el resto de casos hay 3 posibilidades
            - Si la ambulancia tiene exactamente 10 no contagiosos, la heurística devolverá el valor de la distancia al hospital de no contagiosos
            - Si la ambulancia tiene exactamente 9 no contagiosos, la heurística devolverá el valor de la distancia al no contagiosos más lejano más la distancia al hospital de no contagiosos desde la posición del no contagioso más lejano
            - Para el resto de casos se coge el valor de la distancia al enfermo más lejano más la distancia de ese enfermo al su hospital correspondiente
    - Si no quedan enfermos por entregar la heurística devolverá la distancia desde la ambulancia hasta el parking
- **H3**: Esta heurística consta de dos partes
    - Si quedan enfermos por entregar devolverá la distancia del enfermo más lejano más la distancia del hospital correspondiente a este enfermo desde la posición del enfermo más lejano más la distancia desde el hospital al parking
    - Si no quedan enfermos la heurística devolverá la distancia de la ambulancia hasta el parking
- **H4:** Esta heurística consta de varias partes
    - Si quedan enfermos en el mapa se tendrán 3 opciones

        - Si la ambulancia lleva 2 contagiosos y 8 no contagiosos devolverá la distancia al hospital más lejano más la distancia desde ese hospital al parking
        - Si la ambulancia lleva exactamente a 10 no contagiosos devolverá la distancia al hospital de no contagiosos más la distancia desde el hospital al parking
        - Para el resto de casos devuelve suma de la distancia de la ambulancia al enfermo más lejano, la distancia de ese enfermo a su hospital correspondiente y la distancia del hospital al parking
    - Si no quedan enfermos se devolverá la distancia de la ambulancia al parking
- **H5**: Esta heurística busca encontrar un camino desde la posición actual de la ambulancia hasta el enfermo más cercano y se le suma la distancia desde ese enfermo al siguiente más cercano, así hasta recoger a todos los enfermos. Para obtener esta heurística hemos relajado la energía, la prioridad de los enfermos y la capacidad de la ambulancia.

**Resultado de las pruebas**

**Mapa sencillo 3x3**

| **Heurística** | **Solución Encontrada** | **Nodos expandidos** | **Tiempo** | **Coste** | **Longitud del camino** |
|---|---|---|---|---|---|
| **H1** | Si | 308 | 2 ms | 13 | 13 |
| **H2** | Si | 402 | 4 ms | 13 | 13 |
| **H3** | Si | 147 | 1 ms | 13 | 13 |
| **H4** | Si | 147 | 1 ms | 13 | 13 |
| **H5** | Si | 128 | 1 ms | 13 | 13 |

**Mapa con parking encerrado**

| **Heurística** | **Solución Encontrada** | **Nodos expandidos** | **Tiempo** | **Coste** | **Longitud del camino** |
|---|---|---|---|---|---|
| **H1** | No | 1 | 0 ms | X | X |
| **H2** | No | 1 | 0 ms | X | X |

| | | | | | |
|---|---|---|---|---|---|
| **H3** | No | 1 | 0 ms | X | X |
| **H4** | No | 1 | 0 ms | X | X |
| **H5** | No | 1 | 0 ms | X | X |

**Mapa con mucho coste**

| **Heurística** | **Solución Encontrada** | **Nodos expandidos** | **Tiempo** | **Coste** | **Longitud del camino** |
|---|---|---|---|---|---|
| **H1** | Si | 57834 | 7,554 s | 35 | 25 |
| **H2** | Si | 88295 | 10,898 s | 35 | 25 |
| **H3** | Si | 35017 | 2,940 s | 35 | 25 |
| **H4** | Si | 35017 | 3,04 s | 35 | 25 |
| **H5** | Si | 4086 | 0,097 s | 35 | 25 |

**Mapa con solo contagiosos**

| **Heurística** | **Solución Encontrada** | **Nodos expandidos** | **Tiempo** | **Coste** | **Longitud del camino** |
|---|---|---|---|---|---|
| **H1** | Si | 5069 | 69 ms | 26 | 27 |
| **H2** | Si | 4704 | 51 ms | 26 | 27 |
| **H3** | Si | 5069 | 88 ms | 26 | 27 |
| **H4** | Si | 5069 | 66 ms | 26 | 27 |
| **H5** | Si | 887 | 8 ms | 26 | 27 |

**Mapa con solo no contagiosos**

| Heurística | Solución Encontrada | Nodos expandidos | Tiempo | Coste | Longitud del camino |
|---|---|---|---|---|---|
| **H1** | Si | 292 | 3 ms | 14 | 15 |
| **H2** | Si | 292 | 3 ms | 14 | 15 |
| **H3** | Si | 64 | 0 ms | 14 | 15 |
| **H4** | Si | 64 | 0 ms | 14 | 15 |
| **H5** | Si | 20 | 0 ms | 14 | 15 |

**Mapa con contagioso obligatorio**

| Heurística | Solución Encontrada | Nodos expandidos | Tiempo | Coste | Longitud del camino |
|---|---|---|---|---|---|
| **H1** | Si | 116204 | 106 ,395 s | 28 | 29 |
| **H2** | Si | 225432 | 275,42 s | 28 | 29 |
| **H3** | Si | 47047 | 7,971 s | 28 | 29 |
| **H4** | Si | 47047 | 8,605 s | 28 | 29 |
| **H5** | Si | 4433 | 0,112 s | 28 | 29 |

**Mapa recta**

| Heurística | Solución Encontrada | Nodos expandidos | Tiempo | Coste | Longitud del camino |
|---|---|---|---|---|---|
| **H1** | Si | 234 | 1 ms | 24 | 25 |
| **H2** | Si | 227 | 1 ms | 24 | 25 |
| **H3** | Si | 253 | 1 ms | 24 | 25 |
| **H4** | Si | 253 | 1 ms | 24 | 15 |
| **H5** | Si | 122 | 0 ms | 24 | 25 |

**Mapa con enfermo alcanzable pero sin vuelta**

| Heurística | Solución Encontrada | Nodos expandidos | Tiempo | Coste | Longitud del camino |
|---|---|---|---|---|---|
| **H1** | No | 8865 | 87 ms | X | X |
| **H2** | No | 8861 | 81 ms | X | X |
| **H3** | No | 8865 | 76 ms | X | X |
| **H4** | No | 8865 | 70 ms | X | X |
| **H5** | No | 8866 | 72 ms | X | X |

**Prueba de estrés**

Para evaluar el rendimiento general del programa, hemos realizado una prueba en un mapa de mayor tamaño. En un mapa 7x10 se expanden 209515 nodos, tardando un total de 134,42 segundos en ejecutar. Este caso de prueba se encuentra fuera de la carpeta de ASTAR-test, ya que al ejecutarlo con otras heurísticas el tiempo de ejecución sube a horas.

# Conclusiones

Tras analizar cada uno de los mapas podemos obtener que las 5 heurísticas propuestas son admisibles ya que no se sobreestima el coste real y por lo tanto gracias a las características del A* se puede asegurar que se encuentra siempre una solución si existe y además esta solución es la solución óptima del problema. Sin embargo, de las 5 heurísticas podemos observar que no están igual de informadas, ya que difieren en la cantidad de nodos expandidos y por lo tanto en el tiempo de ejecución.
La heurística menos informada es H2, seguida de H1. Tras esto podríamos decir que hay casi un empate entre H3 y H4, aunque H3 tiene a tardar un poco menos en ejecutar, debido a que la heurística es más simple. Pero la heurística más informada que hemos podido obtener es H5, que aunque tenga una alta complejidad nos ayuda a acercarnos mucho más rápido a la solución.
Por esta razón hemos únicamente utilizado H5 para resolver un mapa grande.

# 4. Conclusiones

El desarrollo de esta práctica nos ha permitido afianzar nuestros conocimientos sobre satisfacción de red de restricciones y problemas de búsqueda. Además, nos ha permitido conocer nuevos usos para herramientas ya conocidas como Python y C + +.

En cuanto a la primera parte, Python-constraint es una herramienta útil e intuitiva que permite implementar fácilmente modelos bien definidos. Al desarrollar esta parte, la principal dificultad ha sido modelar correctamente el modelo de forma que tuviésemos el número de soluciones adecuado a la vez que teníamos en cuenta todas las restricciones especificadas por el enunciado.

Sobre la segunda parte, la modelización del problema también ha supuesto trabas al desarrollo. Además, subestimamos en gran medida el coste computacional del A*. Al aumentar el tamaño de los casos de prueba, las ejecuciones han sido cada vez más lentas, haciendo del proceso de prueba del programa un proceso largo y tedioso. También, la definición de heurísticas ha sido complicada, destacando sobre todo la búsqueda de casos límite que volvían nuestras primeras heurísticas no admisibles.

Teniendo todo lo anterior en cuenta, podemos concluir que para problemas de redes de restricciones, la parte maś fundamental es el correcto modelado del problema; y para los problemas de búsqueda informada, la definición de una buena heurística admisible y consistente.